📁 Jirani App - Complete Project Structure Explanation

This document explains **every folder and file** in your Django project and how they work together.

🎯 Understanding Django Architecture

Django follows the **MVT (Model-View-Template)** pattern:

- **Model** = Database structure (what data we store)
- **View** = Logic (what happens when user visits a page)
- **Template** = HTML (what user sees)

📁 Root Level Files

`manage.py`

Purpose: Django's command-line utility
What it does:

- Runs the development server (`python manage.py runserver`)
- Creates migrations (`python manage.py makemigrations`)
- Applies migrations (`python manage.py migrate`)
- Creates superuser (`python manage.py createsuperuser`)

You use it for: All Django administrative commands

`db.sqlite3`

Purpose: Database file
What it does:

- Stores all your data (users, residents, payments, requests, etc.)
- Automatically created when you run migrations
- SQLite is perfect for development (single file database)

Contains tables for:

- Users (Django's built-in)
- Residents
- Payments

- Requests
- Work Orders
- Units
- Announcements

`.gitignore`

Purpose: Tells Git what files NOT to upload to GitHub
What it ignores:

- `__pycache__/` - Python cache files (regenerated automatically)
- `db.sqlite3` - Database (contains sensitive data)
- `*.pyc` - Compiled Python files
- `.env` - Environment variables (passwords, keys)
- `/staticfiles` - Collected static files

Why: Keeps your repository clean and secure

`.gitattributes`

Purpose: Tells GitHub how to classify your code
What it does:

- Marks Python as the primary language
- Excludes CSS/JS from language statistics
- Makes your repo show as a Python project

📁 `jirani_project/` Folder

This is the **main project configuration folder**.

`__init__.py`

Purpose: Makes this directory a Python package
Content: Usually empty
Why it exists: Python requirement

`settings.py`

Purpose: Project-wide configuration

What you configured:

```python
# Installed Apps
INSTALLED_APPS = [
    'django.contrib.admin',      # Admin panel
    'django.contrib.auth',       # User authentication
    'django.contrib.contenttypes', # Content types framework
    'django.contrib.sessions',   # Session management
    'django.contrib.messages',   # Flash messages
    'django.contrib.staticfiles', # Static file handling
    'dashboard',                 # YOUR APP
]

# Database
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

# Static Files
STATIC_URL = '/static/'          # URL prefix for static files
STATIC_ROOT = BASE_DIR / 'staticfiles'  # Where collectstatic puts files
STATICFILES_DIRS = [BASE_DIR / 'dashboard' / 'static']  # Where to find static files

# Media Files
MEDIA_URL = '/media/'            # URL for uploaded files
MEDIA_ROOT = BASE_DIR / 'media'  # Where uploaded files are stored

# Authentication
LOGIN_URL = 'login'              # Where to redirect if not logged in
LOGIN_REDIRECT_URL = 'dashboard' # Where to go after login
LOGOUT_REDIRECT_URL = 'login'    # Where to go after logout
```

Important settings:

- `SECRET_KEY` - Security key (keep secret!)
- `DEBUG = True` - Shows detailed errors (turn off in production)
- `ALLOWED_HOSTS = []` - Which domains can access (add your domain for production)

---

urls.py

Purpose: Main URL routing
What it does: Maps URLs to views

```python
urlpatterns = [
    path('admin/', admin.site.urls),     # Admin panel at /admin/
    path('', include('dashboard.urls')), # Everything else goes to dashboard app
]
```

How it works:

1. User visits `http://127.0.0.1:8000/login/`
2. Django checks this file
3. Sees `''` includes dashboard.urls
4. Passes to `dashboard/urls.py`
5. Finds `login/` matches `views.login_view`
6. Executes that function

---

wsgi.py & **asgi.py**

Purpose: Server gateway interface
What they do: Connect Django to web servers
When used: Deployment to production (Heroku, AWS, etc.)
For development: Not needed

📁 dashboard/ Folder

This is your **main application** - where all the magic happens!

---

models.py

Purpose: Database structure definition
What it contains: Python classes that become database tables

Example:

```python
```

```
class Resident(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    unit_number = models.CharField(max_length=10)
    phone = models.CharField(max_length=15)
    move_in_date = models.DateField()
    is_active = models.BooleanField(default=True)
```

Becomes database table:

```
residents_table
├── id (auto-created)
├── user_id (foreign key to users table)
├── unit_number
├── phone
├── move_in_date
└── is_active
```

Your Models:

1. **Resident** - Tenant information
2. **Payment** - Billing records
3. **Request** - Maintenance requests
4. **WorkOrder** - Work orders with status
5. **Unit** - Property units
6. **Announcement** - Community announcements

Why models are important:

- Django creates database tables automatically
- Provides easy querying: `Resident.objects.filter(unit_number='294')`
- Handles relationships (foreign keys)
- Data validation

---

views.py

Purpose: Business logic (what happens when someone visits a page)
What it contains: Functions that process requests and return responses

Example:

```
python
```

```python
@login_required
def dashboard_view(request):
    # Get the logged-in user's resident profile
    resident = Resident.objects.get(user=request.user)

    # Get their payments
    payments = Payment.objects.filter(resident=resident)

    # Pass data to template
    context = {
        'resident': resident,
        'payments': payments,
    }

    # Render HTML with data
    return render(request, 'dashboard/dashboard.html', context)
```

Flow:

1. User logs in as `john.doe`
2. Visits `/` (dashboard)
3. `dashboard_view()` function runs
4. Queries database for john.doe's data
5. Passes data to HTML template
6. Returns rendered HTML to browser

Your Views:

- `dashboard_view()` - Main dashboard with all data
- `login_view()` - Handles login form
- `logout_view()` - Logs user out
- `signup_view()` - For future registration

Decorators:

- `@login_required` - User must be logged in to access

---

urls.py

Purpose: URL routing for this app
What it contains: Maps URLs to view functions

```python
urlpatterns = [
    path('', views.dashboard_view, name='dashboard'),
    path('login/', views.login_view, name='login'),
    path('logout/', views.logout_view, name='logout'),
]
```

URL Examples:

- `http://127.0.0.1:8000/` → `dashboard_view()`
- `http://127.0.0.1:8000/login/` → `login_view()`
- `http://127.0.0.1:8000/logout/` → `logout_view()`

The `name=` parameter:

- Used in templates: `{% url 'login' %}`
- Makes URLs dynamic and easy to change

`admin.py`

Purpose: Configure Django admin panel
What it does: Registers models so you can manage them in `/admin/`

Should contain:

```python
from django.contrib import admin
from .models import Resident, Payment, Request, WorkOrder, Unit

admin.site.register(Resident)
admin.site.register(Payment)
admin.site.register(Request)
admin.site.register(WorkOrder)
admin.site.register(Unit)
```

Allows you to:

- Add/edit/delete residents
- View all payments
- Manage requests
- Update work orders

- All through a nice web interface at `/admin/`

---

apps.py

Purpose: App configuration
Content: Basic app settings
Usually: Don't need to modify

---

tests.py

Purpose: Automated testing
For future: Write tests to ensure code works correctly
Currently: Can be empty

 **dashboard/migrations/**

Purpose: Database version control
What it contains: Files that track database changes

How it works:

1. You define models in `models.py`
2. Run `python manage.py makemigrations`
3. Django creates migration files (like `0001_initial.py`)
4. Run `python manage.py migrate`
5. Django updates the database

Why migrations are important:

- Track database history
- Can undo changes
- Share database structure with team
- Apply changes safely

Don't delete these files!

 **dashboard/static/**

Purpose: Static files (CSS, JavaScript, Images)

static/css/style.css

Purpose: All styling for your dashboard
What it contains:

- Layout styles (sidebar, main content)
- Component styles (cards, buttons, forms)
- Colors and gradients
- Responsive design (mobile/tablet/desktop)

How Django uses it:

```html
{% load static %}
<link rel="stylesheet" href="{% static 'css/style.css' %}">
```

CSS Organization:

```css
/* Reset and Base Styles */
* { margin: 0; padding: 0; }

/* Sidebar Styles */
.sidebar { width: 280px; background: white; }

/* Card Styles */
.card { background: white; border-radius: 12px; }

/* Responsive Design */
@media (max-width: 768px) { ... }
```

---

static/js/main.js

Purpose: Interactive functionality
What it contains:

- Search functionality
- Animations
- Event listeners
- Chart animations

Examples:

```javascript
// Search contacts
searchInput.addEventListener('input', function(e) {
    // Filter contacts as user types
});

// Notification bell animation
notificationBell.addEventListener('click', function() {
    this.classList.add('ring');
});
```

---

static/images/

Purpose: Store images (logo, building photos, etc.)
Your files:

- logo.png - Jirani App logo
- building.jpg - Building photo for stats card

How to use:

```html
<img src="{% static 'images/logo.png' %}" alt="Logo">
```

---

dashboard/templates/dashboard/

Purpose: HTML templates (what users see)

login.html

Purpose: Login page
What it contains:

- Login form (username, password)
- CSRF token (security)
- Error messages display
- Beautiful gradient design

Django Template Tags:

```html
```

```
{% load static %}          <!-- Load static files -->
{% csrf_token %}           <!-- Security token for forms -->
{% if messages %}          <!-- Show error messages -->
{% for message in messages %}
   {{ message }}
{% endfor %}
{% endif %}
```

---

dashboard.html

Purpose: Main dashboard page

What it contains:

- Sidebar navigation
- Top navigation bar
- Payment summary cards
- New requests display
- Work orders chart
- Delayed orders table
- Vacant units grid

Django Template Variables:

```
html

{{ user.get_full_name }}          <!-- John Doe -->
{{ stats.total_residents }}       <!-- 1 -->
{{ payment_summary.rent.amount }}  <!-- 10335.60 -->

{% for request in new_requests %}
   {{ request.title }}
   {{ request.resident.unit_number }}
{% endfor %}
```

Template Inheritance:

```
html

{% extends 'base.html' %}  <!-- Use base template -->
{% block content %}
   <!-- Your content here -->
{% endblock %}
```

🔄 How Everything Works Together

Example: User Logs In

1. **User visits:** `http://127.0.0.1:8000/login/`
2. **Django checks:** `jirani_project/urls.py` → includes `dashboard/urls.py`
3. **dashboard/urls.py** matches: `path('login/', views.login_view)`
4. **views.py** function runs:

```python
def login_view(request):
    if request.method == 'POST':
        # Get username and password from form
        username = request.POST.get('username')
        password = request.POST.get('password')

        # Check if valid
        user = authenticate(request, username=username, password=password)

        if user is not None:
            login(request, user)
            return redirect('dashboard')  # Go to dashboard
```

5. **If login successful:** Redirects to `dashboard_view()`
6. **dashboard_view() queries database:**

```python
resident = Resident.objects.get(user=request.user)
payments = Payment.objects.filter(resident=resident)
```

7. **Passes data to template:**

```python
context = {'resident': resident, 'payments': payments}
return render(request, 'dashboard/dashboard.html', context)
```

8. **Template renders with data:**

```html
```

```
<div>Unit {{ resident.unit_number }}</div>
<div>Rent: Ksh.{{ payments.rent.amount }}</div>
```

9. **Browser displays:** Beautiful dashboard with user's data
10. **CSS styles it:** `style.css` makes it look beautiful
11. **JavaScript adds interactivity:** Animations, search, etc.

📊 Data Flow Diagram

```
User Browser
    ↓
Django URLs (routing)
    ↓
Views (logic)
    ↓
Models (database queries)
    ↓
Database (SQLite)
    ↓
Models (return data)
    ↓
Views (process data)
    ↓
Templates (HTML + data)
    ↓
User Browser (rendered page)
```

🎓 Key Django Concepts You Used

1. Django ORM (Object-Relational Mapping)

Instead of writing SQL:

```sql
SELECT * FROM residents WHERE unit_number = '294';
```

You write Python:

```python
```

```
Resident.objects.filter(unit_number='294')
```

2. Django Auth System

- User registration/login
- Password hashing
- Session management
- Permission checking

3. Template System

- Variables: `{{ variable }}`
- Tags: `{% for %}` `{% if %}`
- Filters: `{{ text|truncatewords:10 }}`
- Static files: `{% static 'css/style.css' %}`

4. Forms & CSRF Protection

- `{% csrf_token %}` prevents cross-site attacks
- Form validation
- Error handling

5. URL Routing

- Clean URLs: `/dashboard/` not `/index.php?page=dashboard`
- Named URLs for flexibility
- URL parameters

💡 Summary

Python Files:

- `models.py` - What data looks like
- `views.py` - What happens when user visits
- `urls.py` - Which URL goes where
- `admin.py` - Admin panel configuration

Template Files:

- `login.html` - Login page

- `dashboard.html` - Dashboard page
- Uses Django template language

Static Files:

- `style.css` - How it looks
- `main.js` - How it behaves
- `images/` - Pictures

Configuration:

- `settings.py` - Project settings
- `manage.py` - Command line tool
- `.gitignore` - What not to upload

Database:

- `db.sqlite3` - All your data
- `migrations/` - Database history

This structure is standard for **all Django projects**! Once you understand this, you can build any web application! 🚀